## Hyperneat versus rl power for online gait learning in modular robots

D'Angelo, M.; Weel, B.P.M.; Eiben, A.E.

***published in***
Lecture Notes in Computer Science
2014

***DOI (link to publisher)***
10.1007/978-3-662-45523-4_63

***document version***
Publisher's PDF, also known as Version of record

# HyperNEAT versus RL PoWER for Online Gait Learning in Modular Robots

M. D'Angelo[1], Berend Weel[2], and A.E. Eiben[2]

[1] University La Sapienza, Rome, Italy
[2] VU University Amsterdam, The Netherlands
maxxi.d.angelo@gmail.com, b.weel@vu.nl, a.e.eiben@vu.nl

**Abstract.** This paper addresses a principal problem of *in vivo* evolution of modular multi-cellular robots, where robot 'babies' can be produced with arbitrary shapes and sizes. In such a system we need a generic learning mechanism that enables newborn morphologies to obtain a suitable gait quickly after 'birth'. In this study we investigate and compare the reinforcement learning method RL PoWeR with HyperNEAT. We conduct simulation experiments using robot morphologies with different size and complexity. The experiments give insights into the differences in solution quality and algorithm efficiency, suggesting that reinforcement learning is the preferred option for this online learning problem.

**Keywords:** embodied artificial evolution, modular robots, artificial life, online gait learning, reinforcement learning, HyperNEAT

## 1 Introduction

The work described in this paper forms a stepping stone towards the grand vision of embodied artificial evolution (EAE) as outlined in [7]. The essence of this vision is to construct physical systems that undergo evolution 'in the wild', i.e. not in a virtual world inside a computer. There are various possible approaches towards this goal including chemical and biological ones. The one behind this study is based on using a mechatronical substrate, that is, robots.

In general, there are two principal forces behind evolution: selection and reproduction. Selection –at least environmental, objective-free selection– is 'for free' in the real world. Therefore, the main challenge for EAE is reproduction, i.e., the creation of tangible physical artifacts with the ability to reproduce. In our case, this means the need for self-reproducing robots. The approach we follow to this end is based on modular robotics with robotic building blocks capable of autonomous locomotion and aggregation into complex 'multicellular' structures in 3D. In this system evolution will not take place in the morphological space of these pre-engineered modules, but in the morphological space of the multicellular organisms. From the perspective of the multicellular robot bodies the basic robots are merely raw material whose physical properties do not change over time.[1]

[1] Nevertheless, evolving the controllers of these elementary robot modules during the operational period is possible.

In [6] a conceptual framework for systems where robot morphologies and controllers can evolve in real-time and real-space is presented. This framework, dubbed the Triangle of Life, describes a life cycle that does not run from birth to death, but from conception (being conceived) to conception (conceiving one or more children) and it is repeated over and over again, thus creating consecutive generations of 'robot children'. The Triangle of Life consists of 3 stages, Birth, Infancy, and Mature Life, cf. Fig. 1.

In this paper we address a fundamental problem in the Infancy stage. This stage starts when the morphogenesis of a new robot organism is completed and the 'baby robot' is delivered. As explained in [6], the body (morphological structure) and the mind (controller) of such a new organism will unlikely fit each other well. Therefore the new organism needs some fine tuning.This problem –the Control Your Own Body (CYOB) problem– is inherent to evolutionary ALife systems where both bodies and minds undergo changes during reproduction.

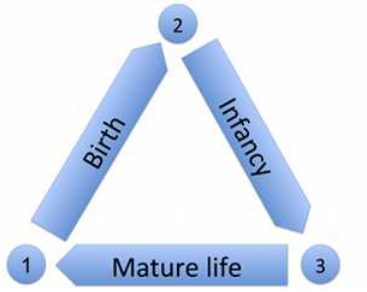



**Fig. 1.** The Triangle of Life. The pivotal moments that span the triangle and separate the 3 stages are: 1) Conception: A new genome is activated, construction of a new organism starts. 2) Delivery: Construction of the new organism is completed. 3) Fertility: The organism becomes ready to conceive offspring.

The work described here addresses the general CYOB problem in a simplified form, by reducing it to gait learning. In the modular robots approach the challenge is to find a method that can learn gaits for all different morphologies that can be created with the given modules and can do this quickly. The problem is highly nontrivial, since a modular robot organism has many degrees of freedom, which leads to a very large search space of possible gaits. Furthermore, this learning process must take place on-the-fly, during the real operational period of the robot organisms. The off-line approach, where a good controller is developed (evolved, learned, hand-coded, etc.) before the robot is deployed is not applicable here, because the life cycle of the Triangle is running in a hands-free mode without being paused for intervention by the experimenter.

In our previous work [5] we have applied a reinforcement learning algorithm PoWeR described by Kober and Peters [14] to solve the CYOB problem and investigated the effects of the shape and size of robot organisms on the performance of the learning method. In this paper we employ an evolutionary approach HyperNEAT [4] that has a good reputation for this type of tasks and compare it with PoWeR. Similarly to [5] we use the learning algorithms with parameter values as recommended by the authors. The grand evolutionary process of the Triangle of Life is not investigated here; it only forms the background context that raises the CYOB problem.

The specific research questions our experiments will try to answer are the following:

1. How do the two approaches compare in terms of the quality of the learned gaits?
2. How do the two approaches compare in terms of the speed of learning?

## 2 Related Work

The design of locomotion for modular robotics is a difficult problem. As explained by Spröwitz: Locomotion requires the creation of rhythmic patterns which satisfy multiple constraints: generating forward motion, without falling over, with low energy, possibly coping with different environments, hardware failures, changes in the environment and/or of the organism [18].

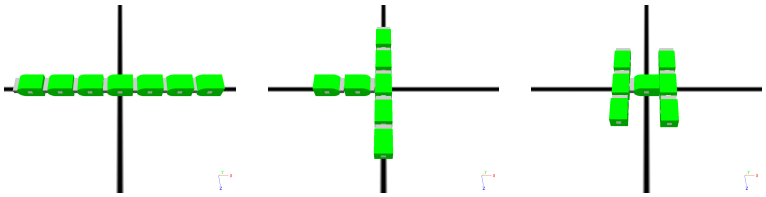
One of the earliest types is gait control tables as in, for instance, [1] and [19]. A gait control table consist of rows of actuator commands with one column for each actuator, each row also has a condition for the transition to the next row.A second major avenue of research is that of neural networks (NN). In particular for locomotion of robot organisms HyperNEAT is used extensively with several studies showing that HyperNEAT is capable of creating efficient gaits for robots [4, 9, 20]. HyperNEAT is discussed in more detail in Section 3. Another successful approach that has received much attention is based on Central Pattern Generators (CPG). CPGs model neural circuitry found in vertebrates which output cyclic patterns without requiring a cyclic input [11]. Each actuator in a robot organism is controlled by the output of a CPG, furthermore the CPGs are connected through certain variables which allows them to synchronise and maintain a certain phase difference pattern. Although sensory input is not strictly needed for CPG's, it can be incorporated to shape the locomotion pattern to allow for turning and modulating the speed. This technique has been shown to produce well performing and stable gaits on both non-modular robots [18, 2] and modular multi-robot organisms [13, 12]. Last, a technique based on artificial hormones has been investigated for the locomotion of modular robot organisms. In this technique artificial hormones are created within robot modules as a response to sensory inputs. These hormones can interact with each other, diffuse to neighbouring modules and act upon output hormones. These output hormones are then used to drive the actuators [10, 17]. Furthermore, some techniques in the field of gait learning employ reinforcement learning algorithms, the specific approaches used can range from Temporal Difference Learning (TDL) to Expectation-Maximization (EM). In TDL one seeks to minimize an error function between estimated and empirical results of a controller, in EM controller parameters are estimated in order to maximize the reward gained using it. These algorithms have been used on modular, e.g. [3] and non modular robots, e.g. [16].

Although there is extensive previous work on this issue, we must stress that, of the techniques described above, only the techniques described in [3], [12] and [18] were actually tested on multiple shapes.

## 3 Experimental Setup

Our primary goal is to compare the reinforcement learning (RL) approach RL PoWER to the population based neural network approach HyperNEAT. Similar to the work in [5] we test both algorithms in various organism morphologies set in a simple environment. These tests are done in simulation using the Webots symulator by Cyberbotics. We use the same adapted YaMoR module [15] as the building blocks for the organisms. The environment for the experiments is an infinite plane, free of obstacles to avoid any extra complexity and the need for supervision. Each experiment starts with the organism lying completely flat at the plane origin.

Nine different robot organisms with different sizes and complexity are defined to examine the generality and scalability of the algorithms. Size and complexity are measured by the number of modules and by the number of extremities, respectively. The experiments are conducted with three complexity levels: organisms with two extremities (I-shape), three extremities (T-shape), and four extremities (H-shape). Each shape is then constructed in three sizes: 7, 11 and 15 modules. A screenshot of the shapes with 7 modules can be seen in Fig. 2, the 11 and 15 module shapes are created by adding modules to the extremities.



(a) Organism I-7 (b) Organism T-7 (c) Organism H-7

**Fig. 2.** Robot organisms of size 7

**RL PoWeR** We use the RL PoWeR reinforcement learning algorithm described by Kober and Peters [14] to optimise the parameters of a set of cyclic splines, called a policy. In such a policy each spline specifies the angular positions of one of the actuator over time. The use of a set of cyclic splines as the representation was taken from [16].

A cyclic spline is a mathematical function that can be defined using a set of $n$ control points. Each control point is defined by $(t_i, \alpha_i)$ where $t_i$ represents time and $\alpha_i$ the corresponding value. $t_i \in [0, 1]$ is defined as

$$t_i = \frac{i}{n-1}, \forall i = 0, \ldots, (n-1) \tag{1}$$

and $\alpha_i \in [0, 1]$ is freely defined, except that the last value is enforced to be equal to the first, i.e. $\alpha_0 = \alpha_n$. These control points are then used for cyclic spline interpolation using the GSL library. Using GSL it is possible to query a spline

for a different number of points than it was defined with, enabling comparison between splines defined with a different number of parameters.

The algorithm starts by creating the initial policy $\pi_0$ with as many splines as there are robots (actuators). The algorithm initialises these splines with $n$ values of 0.5 and then adding Gaussian noise. This initial policy is then evaluated after which it is adapted. This adapted controller is evaluated and adapted again until the stopping condition is reached.

Adaptation is done in two steps which are always applied: Exploitation and Exploration. In the exploitation step, the current splines $\hat{\alpha}$ are optimized based on the outcome of previous controllers, this generates a new set of splines.

$$\hat{\alpha}_{i+1} = \hat{\alpha}_i + \frac{\sum_{j=1}^{k} \hat{\Delta}\alpha_{i,j} R_j}{\sum_{j=1}^{k} R_j} \tag{2}$$

where $\hat{\Delta}\alpha_{i,j}$ represents the difference between the parameters of the i-th policy and j-th policy belonging to a ranking of the best $k$ policies seen so far and $R_j$ its reward. In the exploration phase policies are adapted by applying Gaussian perturbation to the newly generated policy.

$$\hat{\alpha}'_{i+1} = \hat{\alpha}_{i+1} + \hat{\varepsilon}_{i+1},\ \hat{\varepsilon}_{i+1} \sim \mathcal{N}\left(0, \sigma^2\right) \tag{3}$$

where $\hat{\alpha}_{i+1}$ are the parameters after the exploitation step, $\hat{\alpha}'_{i+1}$ the parameters after the exploration step and $\hat{\varepsilon}_{i+1}$ values drawn from a Gaussian distribution with mean 0 and variance $\sigma^2$.

Each controller is evaluated for 23.76 seconds (1,485 time steps) after being used for a recovery period of 3.168 seconds (198 time steps) in order to reduce evaluation noisiness as in [8]. The reward $R$ awarded to a controller $i$ is calculated as follows:

$$R_i = \left( 100 \frac{\sqrt{\Delta_x^2 + \Delta_y^2}}{\Delta_t} \right)^6 \tag{4}$$

where $\Delta_x$ and $\Delta_y$ is the displacement over the $x$ and $y$ axes measured in meters and $\Delta_t$ the time spent in evaluation, as in [16].

The algorithm operating parameters used for the variance and its decay factor are the same as in [16] whereas the others were chosen by hand, without further tuning. Based on our earlier experience, the total number of fitness evaluations was set at 400 and the experiment was repeated for 30 times with organism with different random seeds. An overview of the parameters and the values used in the experiments are described in Table 1.

**HyperNEAT** HyperNEAT is a neuroevolutionary method which evolves a neural network connectivity pattern by using a generative encoding called a CPPN. A CPPN is a network of mathematical functions like Sine, Cosine, Gaussian or Sigmoid, such a network is queried to obtain link weight between nodes of a fixed topology neural network called substrate. An initial population of 25 CPPNs is

randomly generated and each CPPN is queried to obtain a connective pattern for an user defined substrate, the resulting neural network is then evaluated. After having carried out the evaluations the initial population is partitioned into species, within each species the best CPPNs are selected and allowed to mate with their fellows in order to create the next generation. The substrate used in our experiments defines a closed-loop gait and is composed of three layers: input, hidden and output layer. Each layer is a $m \times n$ matrix of neural nodes and its size is calculated as $m = (OrganismSize_x * 2) - 1, n = (OrganismSize_y * 2)$ where $OrganismSize_x$ and $OrganismSize_y$ are the sizes of the organism respectively on the $x$ and $y$ axes measured by the number of modules and the extra column is used for additional user defined inputs. The input layer is fed the angular position of each module servo at the previous time step together with a sine wave function value $s$ defined as $s = \sin(\omega t)$ where $\omega$ represents the maximal angular velocity of the modules servo and $t$ the current time. The output layer produces the angular positions of each module servo for the current time step. The input and output signals are opportunely scaled to and from the interval $[-1, +1]$.

The controller is evaluated for 23.76 seconds (1,485 time steps) with a recovery time between successive evaluations of 3.168 seconds (198 time steps). These times were chosen because they are multiples of the sine wave period and were found to produce better results and avoid organism flipping because of too harsh transitions between gaits.

The fitness of each controller $F_i$ is calculated as in the original experiment [4] and is defined as

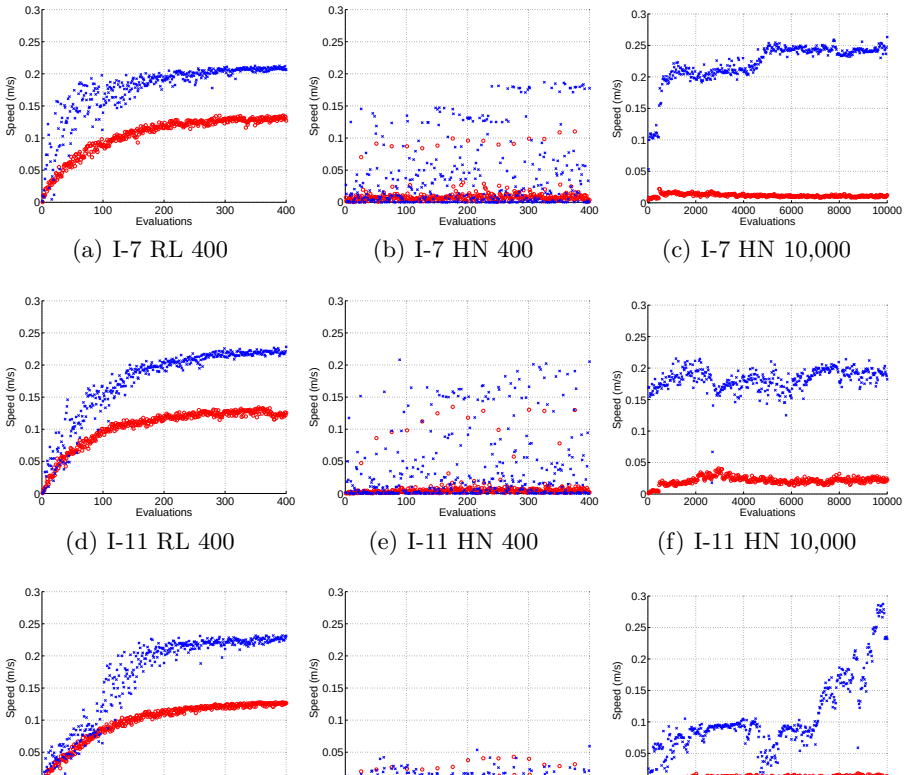
$$F_i = 2^{\left(\Delta_x^2 + \Delta_y^2\right)} \tag{5}$$

where $\Delta_x$ and $\Delta_y$ is the displacement over the $x$ and $y$ axes measured in meters. Note that although the fitness function defined for both methods are different we present our results with a neutral measurement: the speed of the organism in m/s. The fitness function in our view is an integral part of the method and since we use these methods off-the-shelf we also use the fitness function defined in the original paper.

| **Parameter** | **Value** |
|---|---|
| **Common Parameters** | |
| Recovery Steps | 198 |
| Evaluation Steps | 1,485 |
| **HyperNEAT Parameters** | |
| Population Size | 25 |
| Generations | 16 or 400 |
| **RL PoWER Parameters** | |
| Evaluations | 400 |
| Variance | 0.008 |
| Variance Decay | 0.98 |
| Ranking Size | 10 |
| Start Parameters | 2 |
| End Parameters | 100 |

**Table 1.** Experiment Parameters

To make fair comparisons between RL PoWeR and HyperNEAT, the search efforts must be kept equal. The logical way of achieving this is to use the same number of fitness evaluations which was 400 for RL PoWeR. Working with populations of size 25, this implies 16 generations for HyperNEAT. Intuitively, this is a rather small number to get 'decent' evolutionary development. Therefore, we also try another policy, keeping the number of generations equal. This means 400 generations, hence $400 \cdot 25 = 10.000$ fitness evaluations. Note, that the progress




(a) I-7 RL 400 (b) I-7 HN 400 (c) I-7 HN 10,000

(d) I-11 RL 400 (e) I-11 HN 400 (f) I-11 HN 10,000

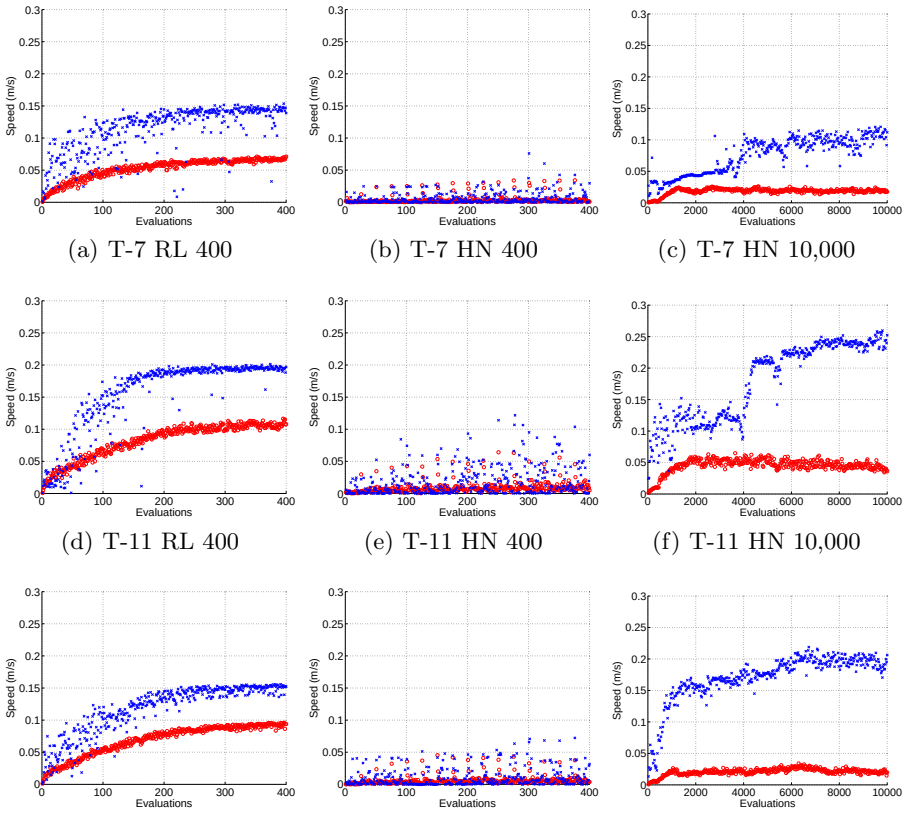
(g) I-15 RL 400 (h) I-15 HN 400 (i) I-15 HN 10,000

**Fig. 3.** Controller performance of RL PoWeR (RL) and HyperNEAT (HN) for the I shape (I-7, I-11, I-15). The $x$ axis represents time measured by the number of evaluations, the $y$ axis shows performance measured by the average speed attained (m/s). The top curve (blue) shows the best single run out of the 30 for RL PoWeR and the HyperNEAT run with 400 evaluations. For HyperNEAT with 10,000 evaluations the top curve shows only the best individuals per generation. The lower curve (red) shows the median speed over 30 runs.

curves plotting fitness in time for RL PoWeR are converging after 400 evaluations, thus the values for 10,000 are the same.

## 4 Experimental Results

The performance of the algorithms is exhibited in Figures 3, 4 and 5 for the I, T and H shape respectively. Each Figure contains 9 plots that show two curves: the lower curve (red in colour prints) displays the median speed of the controllers over 30 runs, the top curve (blue in colour) displays the achieved speeds during




(a) T-7 RL 400 (b) T-7 HN 400 (c) T-7 HN 10,000

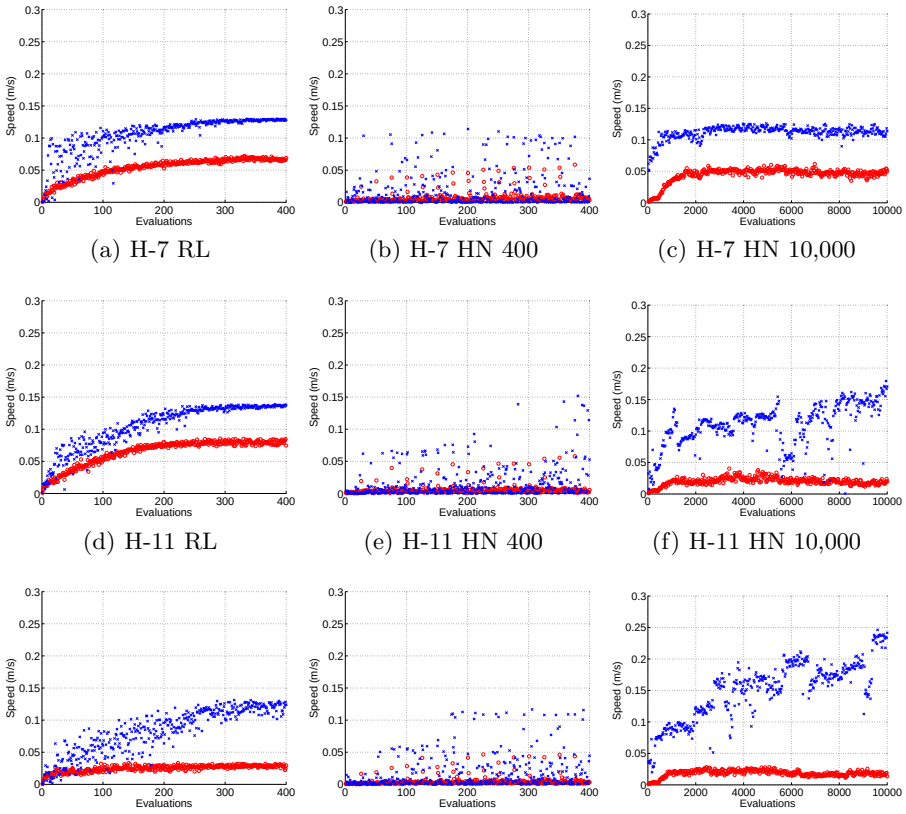
(d) T-11 RL 400 (e) T-11 HN 400 (f) T-11 HN 10,000

(g) T-15 RL 400 (h) T-15 HN 400 (i) T-15 HN 10,000

**Fig. 4.** Controller performance of RL PoWeR (RL) and HyperNEAT (HN) for the T shape (T-7, T-11, T-15). The $x$ axis represents time measured by the number of evaluations, the $y$ axis shows evaluation performance measured by the average speed attained (m/s). The top curve (blue) shows the best single run out of the 30 for RL PoWeR and the HyperNEAT run with 400 evaluations. For HyperNEAT with 10,000 evaluations the top curve shows only the best individuals per generation. The lower curve (red) shows the median speed over 30 runs.

the best run. To improve readability the top curve for HyperNEAT with 10,000 fitness evaluations only shows the performance of the best individual of each generation. The best runs were selected by the performance at the end of the experiment.

Similarly to our previous research we can see that RL PoWeR manages to reach quite a good performance in both the median and best cases with all shapes and sizes we tested. The algorithm converges within 400 evaluations and has a quite stable performance between consecutive trials, i.e. controllers or gaits. Having stable performance through consecutive controllers is an important trait for online learning, where task performance counts from the beginning. A large difference in performance between consecutive gaits implies that poor solutions




(a) H-7 RL (b) H-7 HN 400 (c) H-7 HN 10,000

(d) H-11 RL (e) H-11 HN 400 (f) H-11 HN 10,000

(g) H-15 RL (h) H-15 HN 400 (i) H-15 HN 10,000

**Fig. 5.** Controller performance of RL PoWeR (RL) and HyperNEAT (HN) for the H shape (H-7, H-11, H-15). The $x$ axis represents time measured by the number of evaluations, the $y$ axis shows evaluation performance measured by the average speed attained (m/s). The top curve (blue) shows the best single run out of the 30 for RL PoWeR and the HyperNEAT run with 400 evaluations. For HyperNEAT with 10,000 evaluations the top curve shows only the best individuals per generation. The lower curve (red) shows the median speed over 30 runs.

are being tested as well, and this is clearly disadvantageous for the overall task performance.

The performance of HyperNEAT is quite poor compared to RL PoWeR when using 400 evaluations: even the best controllers in the best run do not reach to the performance of the best run of RL PoWeR in many shapes. 400 evaluations with 25 individuals translates to only 16 generations which is apparently not sufficient. The graphs with 400 evaluations of HyperNEAT also indicate quite large differences in performance of consecutive controllers which is detrimental to the overall performance of the multicellular robot. Running HyperNEAT for 10,000 evaluations, which is 400 generations, leads to much better performance,

| | I | T | H |
|---|---|---|---|
| 7 | 0.10 | 0.04 | 0.02 |
| 11 | 0.09 | 0.05 | 0.04 |
| 15 | 0.10 | 0.06 | 0.02 |

**Table 2.** The table shows the difference in median performance between of RL PoWeR after 400 evaluations and HyperNEAT after 10,000 evaluations. The results of a student's t-test between these median performances showed the difference is significant, the corresponding p values are smaller than 0.01 for each of these tests.

whereas this is not the case for RL PoWeR (plots omitted here). With 10,000 evaluations the best controllers of the best runs of HyperNEAT now sometimes outperform the best controllers by RL PoWeR. The median performance however is still not as good as that of RL PoWeR.

Table 2 shows the difference between the median performance of RL PoWeR after 400 evaluations and HyperNEAT after 10,000. The median performance of RL PoWeR is significantly higher in all cases with a p value smaller than 0.01. This is mainly due to the larger difference in the performance of consecutive controllers with HyperNEAT. This is not surprising, since HyperNEAT works with populations of 25, which means that it does 25 evaluations before applying selection. This causes a more explorative behaviour where several poor solutions are tested too.

Regarding the influence of different body shapes and sizes, we observed the following. The difference in performance is more pronounced for the I shape than for the T and H shapes. The I shape with RL PoWER has 0.90 - 0.10 m/s higher mean performance than HyperNEAT, which is around 30%-34% of the maximum speed achieved by all controllers (the best performance measured was 0.2946 m/s in a run of HyperNEAT I-15). For the T shape the difference in performance is around 0.05 m/s which is roughly 13%-20% of the maximum speed achieved by all controllers. Both these differences are therefore not only statistically significant, but also meaningful. The difference in performance for the H shape is statistically significant, but less pronounced with 7%-13% of the maximum speed.

Considering the speed of learning we can see that RL PoWeR is much faster in reaching a good performance than HyperNEAT. Although the best performance of HyperNEAT with 10,000 evaluations eventually reaches similar performance to RL PoWeR, it uses 25 times as much search effort to this. Even then the median fitness is not much better than that of RL PoWeR at the end of 10,000 evaluations.

## 5 Conclusions

In this paper we addressed the Control Your Own Body problem of *in vivo* evolution of modular multi-cellular robots, where robot 'babies' can be produced with arbitrary shapes and sizes. The problem arises in systems where both morphologies and controllers undergo evolution, such as, for instance our Triangle

of Life framework, because newly created robot organisms can have bodies and controllers that do not fit well. Therefore, every 'baby robot' needs to learn to control its own body quickly by an online learning method, without grace period.

In this study we reduced this to a gait learning problem and investigated two possible learning approaches: The reinforcement learning algorithm RL PoWeR and the neuro-evolutionary approach HyperNEAT. We took 'off-the-shelf' implementations of these algorithms and conducted simulation experiments on a predefined testbed of robot morphologies with 3 different sizes and 3 levels of complexity.

Regarding the quality of learned gaits we have found that RL PoWeR –that iterates only one single solution– reaches quite reasonable speeds in the median case and good speeds in the best case. HyperNEAT on the other hand seems encumbered by the population of 25 and cannot equal the performance of RL PoWeR within 400 evaluations. After 10,000 evaluations there are some runs that are able to outperform RL PoWeR when looking at the best controller of the best run. However, the median is still much lower than RL PoWeR, because HyperNEAT does more exploration than RL PoWeR, which leads to a larger difference in performance between consecutive controllers. This leads to a lower overall task performance, which is undesirable in online learning.

With regards to the speed of the algorithms, we can see that RL PoWeR is much faster in achieving a high performance than HyperNEAT. Surprisingly this quick convergence does not seem to come at the cost of solution quality as one would expect. To conclude, the main finding of our research is that the RL PoWeR algorithm is the preferable over HyperNEAT for on-line learning.

Further work will be carried out along several lines. First we want to tune the parameters for both the RL PoWER and HyperNEAT algorithms on this problem to improve their performances. Furthermore we will investigate the algorithms stability with regards to failed modules and other disasters. Finally, we would like to validate these results by replicating the experiments using real hardware.

## References


1. Bongard, J., Zykov, V., Lipson, H.: Resilient machines through continuous self-modeling. Science 314(5802), 1118–1121 (2006)
2. Christensen, D.J., Larsen, J.C., Støy, K.: Fault-tolerant gait learning and morphology optimization of a polymorphic walking robot. Evolving Systems (2013)
3. Christensen, D.J., Schultz, U.P., Støy, K.: A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots. Robotics and Autonomous Systems 61(9), 1021–1035 (2013)
4. Clune, J., Beckmann, B.E., Ofria, C., Pennock, R.T.: Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In: IEEE Congress on Evolutionary Computation (CEC) 2009. pp. 2764–2771. IEEE Press (2009)
5. D'Angelo, M., Weel, B., Eiben, A.: Online gait learning for modular robots with arbitrary shapes and sizes. In: Proceedings of the 2nd International Conference on the Theory and Practice of Natural Computing, TPNC 2013. vol. LNCS 8273 (2013)

6. Eiben, A.E. et al. , Bredeche, N., Hoogendoorn, M., Stradner, J., Timmis, J., Tyrrell, A., Winfield, A.: The triangle of life: Evolving robots in real-time and real-space. In: Lió, P., Miglino, O., Nicosia, G., Nolfi, S., Pavone, M. (eds.) Advances in Artificial Life, (ECAL) 2013. pp. 1056–1063. MIT Press (2013)
7. Eiben, A.E., Kernbach, S., Haasdijk, E.: Embodied artificial evolution. Evolutionary Intelligence 5(4), 261–272 (2012)
8. Haasdijk, E., Eiben, A.E., Karafotias, G.: On-line evolution of robot controllers by an encapsulated evolution strategy. In: IEEE Congress on Evolutionary Computation (CEC) 2010. pp. 1–7. IEEE Press (2010)
9. Haasdijk, E., Rusu, A.A., Eiben, A.E.: HyperNEAT for locomotion control in modular robots. In: Hornby, G.S., Sekanina, L., Haddow, P.C. (eds.) Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, vol. 5216, pp. 169–180. Springer (2010)
10. Hamann, H., Stradner, J., Schmickl, T., Crailsheim, K.: A hormone-based controller for evolutionary multi-modular robotics: From single modules to gait learning. In: IEEE Congress on Evolutionary Computation (CEC) 2010. pp. 1–8. IEEE Press (2010)
11. Ijspeert, A.J.: Central pattern generators for locomotion control in animals and robots: A review. Neural Networks 21(4), 642–653 (2008)
12. Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K., Kokaji, S.: Automatic locomotion design and experiments for a modular robotic system. IEEE/ASME Transactions on Mechatronics 10(3), 314–325 (2005)
13. Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Kokaji, S., Murata, S.: Distributed adaptive locomotion by a modular robotic system, M-TRAN II. In: Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2004. vol. 3, pp. 2370–2377. IEEE Press (2004)
14. Kober, J., Peters, J.: Learning motor primitives for robotics. In: IEEE International Conference on Robotics and Automation (ICRA) 2009. pp. 2112–2118. IEEE Press (2009)
15. Möckel, R., Jaquier, C., Drapel, K., Dittrich, E., Upegui, A., Ijspeert, A.: YaMoR and Bluemove – an autonomous modular robot with Bluetooth interface for exploring adaptive locomotion. In: Tokhi, M.O., Virk, G., Hossain, M.A. (eds.) Proceedings of the 8th International Conference on Climbing and Walking Robots (CLAWAR) 2005, pp. 685–692. Springer (2006)
16. Shen, H., Yosinski, J., Kormushev, P., Caldwell, D.G., Lipson, H.: Learning fast quadruped robot gaits with the RL PoWER spline parameterization. Cybernetics and Information Technologies 12(3), 66–75 (2012)
17. Shen, W.M., Salemi, B., Will, P.: Hormones for self-reconfigurable robots. In: Pagello, E. et al (eds.) Proceedings of the 6th International Conference on Intelligent Autonomous Systems (IAS-6). pp. 918–925. IOS Press (2000)
18. Spröwitz, A., Moeckel, R., Maye, J., Ijspeert, A.J.: Learning to move in modular robots using central pattern generators and online optimization. The International Journal of Robotics Research 27(3-4), 423–443 (2008)
19. Yim, M.: A reconfigurable modular robot with many modes of locomotion. In: Proceedings of International Conference on Advanced Mechatronics. pp. 283–288. Japan Society of Mechanical Engineers, Tokio, Japan (1993)
20. Yosinski, J., Clune, J., Hidalgo, D., Nguyen, S., Zagal, J., Lipson, H.: Evolving robot gaits in hardware: the HyperNEAT generative encoding vs. parameter optimization. In: Lenaerts, T., Giacobini, M., Bersini, H., Bourgine, P., Dorigo, M., Doursat, R. (eds.) Advances in Artificial Life, (ECAL) 2011. pp. 890–897. MIT Press (2011)